Temario Iniciación al AngularJs

1. Conceptos

Directiva :El concepto de directiva es uno de los más complicados y hace referencia a un conjunto de atributos a nivel de HTML que Angular define como un DSL (Domain Specific Language). Esto permite extender las capacidades que HTML. Ejemplos de estas directivas son : ng-click, ng-repeat, ng-app, ng-controller etc. Cada una aporta su propia funcionalidad.

Controller :Se trata del concepto clásico de controlador que realiza tareas de comunicación entre la vista y el modelo.

Scope :El scope es un objeto que comparte información entre la vista y el controlador, permitiendo al controlador asignar nuevos objetos y funciones a la vista.

View :Este es el concepto más sencillo de entender ya que hace referencia a un conjunto de etiquetas HTML que conforman la vista que se muestra al usuario. Angular usa su propio motor de plantillas para gestionarlas.

Servicios:Los servicios se encargan de la parte del modelo y permiten diseñar un API consistente para trabajar con el servidor.

Filtros: Los filtros apoyan a la vista a la hora de presentar los datos realizando transformaciones sobre ellos.

**2. Un primer ejemplo**

(ver ejemplo\_iniciacion.html)

Expliquémos el archivo y su funcionamiento:

La directiva ng-app le dice a AngularJS que el <html> es el dueño de la aplicación.

La directiva ng-model enlaza el valor del input con la variable de la aplicación “name”

La directiva ng-bind enlaza el contenido del <p> con la variable de la aplicación name. Tb se puede hacer usando {{}}

**3. Iniciación a las Directivas**

Las directivas son atributos que se le añaden al código HTML con el prefijo ng.

La directiva ng-init inicializa variables de la aplicación Angular

<div ng-app="" ng-init="firstName='John'">  
  
<p>The name is <span ng-bind="firstName"></span></p>  
  
</div>

SI queremos que las directivas sean tomadas como valores válidos de atributos HTML, podemos usar data-ng:

<div data-ng-app="" data-ng-init="firstName='John'">  
  
<p>The name is <span data-ng-bind="firstName"></span></p>  
  
</div>

**4. Iniciación a las Expresiones**

Se escriben entre {{}}. La salida será exactamente lo que la expresión quiere decir (ver ejemplo\_expresiones.html)

Las expresiones en AngularJS se pueden linkar con datos Angular en el HTML de la misma manera que la directiva ng-bind

(ver ejemplo\_expresiones2.html)

AngularJS expressions bind AngularJS data to HTML the same way as the ng-bind directive.

**5. Iniciación a las Aplicaciones**

Los módulos definen las aplicaciones. Los controladores controlan las aplicaciones. La directiva ng-app define la aplicación, la directiva ng-controller define el controlador.

(ver ejemplo\_aplicacion.html)

EL módulo define la aplicación:

var app = angular.module('myApp', []);

El controlador controla la aplicación

app.controller('myCtrl', function($scope) {  
 $scope.firstName= "John";  
 $scope.lastName= "Doe";  
});

**Expresiones en AngularJS**

Las expresiones en AngularJS se escriben dentro de dobles llaves {{}}. También pueden ser escritas dentro de una directiva ng-bind=”expresión”. Angular resolverá la expresión y devolverá el valor exactamente donde la expresión se escribe. Pueden contener cadenas, operadores y variables.

Por ejemplo: {{ 5 + 5 }} or {{ firstName + " " + lastName }}

<!DOCTYPE html>  
<html>  
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>  
<body>  
  
<div ng-app="">  
 <p>My first expression: {{ 5 + 5 }}</p>  
</div>  
  
</body>  
</html>

Si le quitamos la directiva ng-app, qué sucederá?

Podemos escribir expresiones donde querramos, AngularJS las resolverá y devolverá el resultado. Por ejemplo, podemos cambiar las propiedades del CSS de la página

<div ng-app="" ng-init="myCol='lightblue'">  
  
<input style="background-color:{{myCol}}" ng-model="myCol">  
  
</div>

Los números se tratan como en Javascript:

<div ng-app="" ng-init="quantity=1;cost=5">  
  
<p>Total in dollar: {{ quantity \* cost }}</p>  
  
</div>

Lo mismo usando ng-bind:

<div ng-app="" ng-init="quantity=1;cost=5">  
  
<p>Total in dollar: <span ng-bind="quantity \* cost"></span></p>  
  
</div>

Las cadenas también se tratan como en Javascript:

<div ng-app="" ng-init="firstName='John';lastName='Doe'">  
  
<p>The name is {{ firstName + " " + lastName }}</p>  
  
</div>

Si usamos ng-bind:

<div ng-app="" ng-init="firstName='John';lastName='Doe'">  
  
<p>The name is <span ng-bind="firstName + ' ' + lastName"></span></p>  
  
</div>

Los objetos, también se tratan como en Javascript:

<div ng-app=" ng-init="person={firstName:'John',lastName:'Doe'}">  
  
<p>The name is {{ person.lastName }}</p>  
  
</div>

Si usamos ng-bind:

<div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}">  
  
<p>The name is <span ng-bind="person.lastName"></span></p>  
  
</div>

Y los arrays:

<div ng-app="" ng-init="points=[1,15,19,2,40]">  
  
<p>The third result is {{ points[2] }}</p>  
  
</div>

Con ng-bind:

<di  ng-app="" ng-init="points=[1,15,19,2,40]">  
  
<p>The third result is <span ng-bind="points[2]"></span></p>  
  
</div>

**Módulos**

Un módulo define una aplicación. El módulo es un contenedor de las diferentes partes de la aplicación. El módulo es un contenedor para los controladores de la aplicación. Los controladores siempre pertenecen a un módulo.

El módulo se crea usando la instrucción angular.module

<div ng-app="myApp">...</div>  
  
<script>  
  
var app = angular.module("myApp", []);  
  
</script>

El parámetro “myApp” se refiere a un elemento HTML en el cual la aplicación correrá. Ahora podemos añadir controladores, directivas, filtros y más a la aplicación Angular

Para añadir un controlador a una aplicación, y referirnos al controlador, usaremos la directiva ng-controller:

<div ng-app="myApp" ng-controller="myCtrl">  
{{ firstName + " " + lastName }}  
</div>  
  
<script>  
  
var app = angular.module("myApp", []);  
  
app.controller("myCtrl", function($scope) {  
 $scope.firstName = "John";  
 $scope.lastName = "Doe";  
});  
  
</script>

(ver ejemplo\_controlador.html)

**Añadir una directiva**

Angular tiene una serie de directivas que podemos usar para añadir funcionalidades a nuestra aplicación. Además, puedes añadir tus propias directivas

(ver ejemplo\_directiva.html)

**Módulos y controladores en ficheros**

Lo normal es que los módulos y los controladores se escriban en ficheros aparte.

(ver ejemplo\_archivos\_externos.html)

**Directivas en AngularJS**

AngularJS te permite extender el HTML con nuevos atributos llamados Directivas. AngularJS tiene un conjunto de directivas predefinidas que ofrecen funcionalidad a tus aplicaciones. AngularJS también te permite definir tus propias directivas.

Las directivas de AngularJS que extienden el HTML comienzan con el prefijo ng-. La directiva ng-app inicializa la aplicación en AngularJS. La directiva ng-init inicializa los datos de la aplicación. La directiva ng-model enlaza el valor de los controles HTML (input, select, textarea) a los datos de la aplicación.

(ver ejemplo\_directivas.html)

La directiva ng-app también le dice a AngularJS que el div es el dueño de la aplicación en AngularJS.

La expresion {{ firstName }} en el ejemplo anterior, es una expresión de data binding de AngularJS. El data binding en AngularJS enlaza expresiones con datos. {{ firstName }} enlaza con ng-model="firstName".

(ver ejemplo\_directivas2.html)

**Repitiendo elementos HTML**

La directiva ng-repeat repite un elemento HTML. Clona elementos HTML para cada uno de los items de la colección. También podemos usarla con un array de objetos:

(ver ejemplo\_ng\_repeat2.html)

**La directiva ng-app**

Esta directiva define el elemento raiz de la aplicación Angular. Además, inicializa automáticamente la aplicación cuando la página web termina de cargarse.

**La directiva ng-init**

Define los valores iniciales para una aplicación Angular. Normalmente, no usaremos esta directiva. En su lugar, usaremos un controlador o un módulo.

**La directiva ng-model**

Esta directiva enlaza el valor de un control de HTML con los datos de la aplicación. También puede:

a) Proporcionar validación de tipo para los datos de la aplicación (nombre, email, required)

b) Proporcionar status para los datos de la aplicación ( inválidos, sucios, error)

c) Proporcionar clases a elementos HTML

d) Enlazar elementos HTML a formularios HTML

**Crear nuevas directivas**

Además de las directivas propias de AngularJS, podemos crear nuestras propias directivas. Podemos hacerlo usando la función .directive. Para invocar la nueva directiva, debemos construir un elemento HTML con el mismo valor de etiqueta que la nueva directiva. Cuando nombremos una directiva, usaremos el nombrado camelcase, como por ejemplo w3TestDirective, pero cuando la invoquemos, debemos usar el nombre separado por guiones medios, w3-test-directive.

(ver ejemplo\_nueva\_directiva.html)

Se puede invocar una nueva directiva usando el nombre de un elemento, un atributo, una clase o un comentario.

El ejemplo siguiente produce el mismo resultado:

<div w3-test-directive></div> (ver ejemplo\_nueva\_directiva3.html)

<div class="w3-test-directive"></div> (ver ejemplo\_nueva\_directiva2.html)

<!-- directive: w3-test-directive → (ver ejemplo\_nueva\_directiva4.html)

**La directiva ng-model**

Esta directiva enlaza el valor de un control HTML con los datos de una aplicación. Es decir, nos puede permitir enlazar el valor de un input de un formulario con una variable creada por AngularJSAngularJS

(ver directiva\_ng\_model.html)

**Two Way Binding**

El enlazado puede funcionar en los dos sentidos. Si el usuario cambia el valor del input, la propiedad de AngularJS cambiará también su valor

(ver two\_way\_binding.html)

La directiva ng-model puede proporcionar validación para el tipo de dato (número, email, required):

(ver validacion\_email.html)

En el ejemplo anterior, el spam se visualizará sólo si la expresión en ng-show retorna un valor válido.

En el siguiente ejemplo, validará el status de los datos de la aplicación ( válido, sucio, tocado, error)

(ver validacion\_status.html)

La directiva ng-model proporciona también clases para el HTML, dependiendo de su status.

(ver validacion\_css.html)

La directiva ng-model añade/quita las siguientes clases, de acuerdo al status del campo del formulario: ng-empty, ng-not-empty, ng-touched, ng-untouched, ng-valid, ng-invalid, ng-dirty, ng-pending, ng-pristine.

**Data Binding (enlazado de datos)**

El Data Binding en AngularJS es la sincronización entre el modelo y la vista. Las aplicaciones en AngularJS usualmente tienen un modelo de datos. El modelo de datos es una colección de datos accesible a la aplicación.

Va  app = angular.module('myApp', []);  
app.controller('myCtrl', function($scope) {  
 $scope.firstname = "John";  
 $scope.lastname = "Doe";  
});

El contenedor HTML donde la aplicación AngularJS es mostrada, se denomina vista (view). La vista tiene acceso al modelo, y hay distintas maneras de mostrar el modelo de datos en la vista. Puedes usar la directiva ng-bind la cual enlazará el innerHTML del elemento con la propiedad del modelo especificada.

(ver data\_binding.html)

También puedes usar {{ }} para mostrar el contenido de un modelo:

(ver data\_binding2.html)

O también puedes usar la directiva ng-model en el control HTML para enlazar la vista con el modelo

(ver directiva\_ng\_model2.html) Esta directiva proporciona el two way binding. Ya sabemos que consiste en la sincronización entre la vista y los datos del modelo. Cuando los datos del modelo cambian, la vista refleja el cambio, y cuando los datos cambian en la vista, el modelo es actualizado también. Esto sucede inmediata y automáticamente, lo cual asegura que el modelo y la vista se actualizan a la vez.

(ver directiva\_ng\_model3.html)

Debido a la sincronización inmediata del modelo y la vista, el controlador puede ser completamente separado de la vista, y simplemente concentrarte en el modelo. Gracias al data binding en AngularJS, la vista reflejará cualquier cambio en el modelo.

(ver directiva\_ng\_model4.html)

**Controladores**

Los controladores controlan los datos de las aplicaciones AngularJS. Son objetos regulares de Javascript. La directiva ng-controller define el controlador de la aplicación. Un controlador es un objeto constructor de Javascript.

(ver controladores1.html)

Explicación del código:

La aplicación AngularJS es definida por ng-app=myApp. La aplicación corre dentro de un div. El atributo ng-controller=myCtrl es una directiva de AngularJS. Define un controlador. La función myCtrl es una función Javascript.

AngularJS invocará el controlador con un objeto $scope. En AngularJS, $scope es el objeto de la aplicación ( el dueño de la aplicación y las funciones ). El controlador crea dos propiedades ( variables ) en el scope ( firstName y lastName ) . La directiva ng-model enlaza el campo input a las propiedades del controlador ( firstName y LastName)

El ejemplo anterior muestra un controlador con dos propiedades: firstName y lastName. Un controlador puede tener métodos ( funciones como variables )

(ver controladores2.html)

En aplicaciones de gran tamaño, lo normal es almacenar los controladores en archivos externos. En el anterior ejemplo, podemos cortar el código entre las etiquetas script y almacenarlo en un archivo personController.js, al que llamaremos desde el código HTML

<script src="personController.js"></script>

Veamos otro ejemplo:

(ver controladores3.html)

**Scopes**

El scope es la parte enlazante entre el HTML (la vista) y el Javascript (el controlador). El scope es un objeto con sus propiedades y métodos. Es accesible tanto para la vista como para el controlador.

**Cómo usar el scope?**

Cuando creamos un controlador en AngularJS, pasamos el objeto $scope como parámetro:

(ver scope1.html)

Cuando añadimos propiedades al $scope, en el controlador, la vista (HTML) tiene acceso a esas propiedades. En la vista, no usamos el prefijo $scope, sólo nos referimos al nombre de la propiedad, como en el caso de {{ carname}}

Si consideramos que una aplicación AngularJS consiste en una vista (HTML), un modelo que consiste en todos los datos accesibles por la aplicación, y un controlador, que es la función Javascript que crea/cambia/borra/controla los datos, entonces el scope es el modelo. El scope es un objeto Javascript con propiedades y métodos, los cuales son accesibles desde la vista y el controlador.

(ver scope2.html)

Es importante conocer con cual scope estás tratando en cada momento. En los dos ejemplos anteriores, sólo hay un scope, por lo tanto conocer tu scope no es un problema. Pero para aplicaciones más grandes podemos tener secciones en el HTML las cuales tienen sólo acceso a ciertos scopes.

(ver scope3.html)

Cada elemento li tiene acceso al objeto repetido actual, en este caso una cadena, la cual es referida usando “x”.

**Root Scope**

Todas las aplicaciones tienen un $rootScope, el cual es el scope creado en el elemento HTML que contiene la directiva ng-app. El rootScope es accesible desde toda la aplicación. Si la variable tiene el mismo nombre en ambos scopes, el actual y el root, la aplicación usa el scope actual.

(ver rootscope.html)

**Filtros**

Los filtros son añadidos en AngularJS para formatear los datos. AngularJS proporciona filtros para transformar los datos:

currency

date

filter

json

limitTo

lowercase

number

orderBy

uppercase

Los filtros pueden ser añadidos a las expresiones usando el carácter pipe |, seguido de un filtro. El filtro de formato uppercase formatea cadenas a mayúsculas:

(ver filtros1.html)

El filtro lowercase formatea cadenas a minúsculas:

<div ng-app="myApp" ng-controller="personCtrl">  
  
<p>The name is {{ lastName | lowercase }}</p>  
  
</div>

Los filtros se pueden añadir a directivas, como ng-repeat, usando el carácter pipe | seguido de un filtro:

(ver filtros2.html)

El filtro currency formatea un número como moneda:

(ver filtros3.html)

El filtro filter filtra un subconjunto de un array. Puede ser usado sólo en arrays, y devuelve un array conteniendo sólo los items que coinciden con el filtro.

(ver filtros4.html)

Podemos cambiar el orden de una tabla, haciendo click en la cabecera:

(ver filtros5.html)

Añadiendo la directiva ng-click, en la cabecera de la tabla, podemos correr una función que cambie el orden del array.

También puedes crear tus propios filtros, registrando una función factory de filtrado con tu módulo. En este caso, ver filtros6.html, el filtro myFormat cambiará a mayúsculas los caracteres impares.

# Servicion AngularJS

En AngularJS, podemos usar sus propios servicios o crear los nuestros propios.

## Qué es un servicio?

En AngularJS, un servicio es una función o un ibjeto que está accesible y limitado por nuestra aplicación AngularJS. Angular JS tiene unos 30 servicios predefinidos. Uno de ellos es el servicio $location.

El servicio $location tiene un método que retorna la información acerca de la localización de la página web actual.

(ver servicios1.html)

Hay que notar que el servicio $location es pasado al controlador como argumento. En order de usar el servicio en el controlador, debe ser definido como una dependencia.

## Por qué usar servicios?

Para muchos de los servicios, cono el $location, parece como si pudieras usar objetos que están en el DOM, como el window.location y que podrías, pero esto tiene algunas limitaciones, al menos para tus aplicaciones AngularJS.

AngularJS supervisa constantemente tu aplicación, y, para que maneje los cambios y eventos correctamente, AngularJS prefiere que use el servicio $location en lugar del objeto ventana.ubicación.

## El servicio $http

El servicio $http es uno de los servicios más utilizados en las aplicaciones AngularJS. El servicio realiza una solicitud al servidor y permite que su aplicación maneje la respuesta.

(ver servicios2.html)

El servicio $timeout

El servicio $timeout es la version AngularJS de la función window.setTimeout

(ver servicios3.html)

El servicio $interval

El servicio $interval es la versión AngularJS de la función wondow.setInterval

(ver servicios4.html)

Crear tu propio servicio

Para crear tu propio servicio, debes conectar el servicio al módulo

app.service('hexafy', function() {  
 this.myFun = function (x) {  
 return x.toString(16);

}  
});

Para usar tu propio servicio, añádelo como dependencia cuando defines el controlador:

app.controller('myCtrl', function($scope, hexafy) {  
 $scope.hex = hexafy.myFunc(255);  
});

(ver servicios5.html)

**Usar tu propio servicio dentro de un filtro**

Una vez que hayas creado el servicio y lo hayas conectado a la aplicación, puedes usarlo en cualquier controlador, directiva, filtro o incluso dentro de otro servicio. Para usar el servicio dentro de un filtro, agrégalo como dependencia al definir el filtro.

(ver servicios6.html)

Puedes usar el filtro cuando muestres valores desde un objeto o un array:

(ver servicios7.html)

**AngularJS y AJAX - $http**

$http es un servicio de AngularJS que permite leer datos desde servidores remotos. El servicio $http hace una petición a un servidor, el cual devuelve una respuesta.

(ver servicio\_http1.html)

**Métodos**

El ejemplo anterior usa el método GET. El método GET es un método abreviado del servicio $http. Hay otras abreviaturas o métodos del servicio $http:

.delete()

.get()

.head()

.jsonp()

.patch()

.post()

.put()

(ver servicio\_http2.html)

Los ejemplos anteriores ejecutan el servicio $http con un objeto como argumento. El objeto especifica el método HTTP, la URL, qué hacer en caso de éxito y qué hacer en caso de fallo.

**Propiedades**

La respuesta del servidor es un objeto con las siguientes propuedades:

.config the object used to generate the request.

.data a string, or an object, carrying the response from the server.

.headers a function to use to get header information.

.status a number defining the HTTP status.

.statusText a string defining the HTTP status.

(ver servicio\_http3.html)

Para manejar los errores, añadimos más funciones al método .then

(ver servicio\_http4.html)

**Json**

Se espera que los datos que obtenga de la respuesta estén en formato JSON.

JSON es una excelente manera de transportar datos y es fácil de usar dentro de AngularJS o cualquier otro JavaScript.

Ejemplo: en el servidor tenemos un archivo que devuelve un objeto JSON que contiene 15 clientes, todos envueltos en una matriz llamada registros.

(ver servicio\_http5.html)

Aplicación explicada:

La aplicación define el controlador clientsCtrl, con un objeto $scope y $http.

$http es un objeto XMLHttpRequest para solicitar datos externos.

$http.get() lee datos JSON de https://www.w3schools.com/angular/customers.php.

En caso de éxito, el controlador crea una propiedad, myData, en el ámbito, con datos JSON del servidor.

**AngularJS y las tablas**

Mostrar tablas con AngularJS es muy sencillo.

( ver tablas.html)

Para ordenar la tabla, le añadimos un filtro orderBy

(ver tablas2.html)

Podemos añadir el índice a la tabla usando un <td> que contenga $index

(ver tablas3.html)

**AngularJS y los selects**

AngularJS te permite crear listas dropdown basadas en un array o en un objeto. Si quieres crear una lista dropdown, basada en un objeto o en un array, deberías usar la directiva ng-option

(ver select.html)

También podemos usar la directiva ng-repeat en lugar de la ng-option, porque ng-repeat repite un bloque de HTML. Pero la directiva ng-options fue creada especialmente para rellenar una lista dropdown con opciones.

(ver select2.html)

Si tenemos un array de objetos

$scope.cars = [

{model : "Ford Mustang", color : "red"},

{model : "Fiat 500", color : "white"},  
 {model : "Volvo XC90", color : "black"}  
];

( ver select3.html) Cuando usamos el valor como un objeto, hay que usar ng-value en lugar del valor

( ver select4.html)

Cuando el valor seleccionado es un objeto, puede guardar más información, y tu aplicación será más flexible. Usaremos ng-options en ese caso. Asumamos que tenemos un objeto

$scope.cars = {  
 car01 : "Ford",  
 car02 : "Fiat",  
 car03 : "Volvo"  
};

( ver select5.html ) El valor seleccionado será siempre el value en el par key-value. El value en el par key-value puede ser también un objeto. Por ejemplo:

$scope.cars = {  
 car01 : {brand : "Ford", model : "Mustang", color : "red"},  
 car02 : {brand : "Fiat", model : "500", color : "white"},  
 car03 : {brand : "Volvo", model : "XC90", color : "black"}  
};

( ver select6.html)

**AngularJS y SQL**

AngularJS es perfecto para mostrar datos de una base de datos. Sólo hay que asegurarse de recibir los datos en un json.

( ver customer\_mysql.php )

( ver mysql.html )

**AngularJS y el HTML DOM**

AngularJS tiene directivas que linkan los datos de la aplicación con elementos del DOM del HTML. Por ejemplo, la directiva ng-disabled linka datos de la aplicación al atributo disabled de algunos elementos del HTML

( ver dom.html)

La directiva ng-show muestra o oculta elementos del HTML. ( ver dom2.html ) Esta directiva muestra o oculta elementos basado en el valor de ng-show. Puedes usar una expresión que sea evaluada como true o false ( ver dom3.html )

**AngularJS y los eventos**

Puedes añadir listeners AngularJS a tus elementos HTML usando una o más de las siguientes directivas: ng-blur, ng-change, ng-click, ng-copy, ng-cut, ng-dblclick, ng-focus, ng-keydown, ng-keypress, ng-keyup, ng-mousedown, ng-mouseenter, ng-mouseleave, ng-mousemove, ng-mouseover, ng-mouseup, ng-paste

Las directivas de eventos permiten correr funciones AngularJS en ciertos eventos del usuario. Un evento AngularJS no sobreescribe un evento HTML, ambos serán ejecutados.

**Eventos de ratón**

Un evento de ratón ocurre cuando el ratón se mueve sobre un elemento, en este orden:

ng-mouseover, ng-mouseenter, ng-mousemove, ng-mouseleave

O cuando el botón de un ratón se clicka sobre algún elemento, en este orden:

ng-mousedown, ng-mouseup, ng-click

( ver eventos.html )

La directiva ng-click define un código que será ejecutado cuando el elemento sea clicado

( ver eventos2.html )

( ver eventos3.html) La variable showMe comienza con el valor false. La función myFunc cambia el valor de showMe usando el operador not !

También podemos pasar el objeto $event como argumento cuando usemos una función. El objeto $event contiene el objeto sobre el que se ha desarrollado el evento.

**Formularios**

Veamos un pequeño ejemplo de validación de formularios

( ver formulario.html )

La directiva ng-model linka los elementos del input con el modelo. El objeto modelo tiene dos propiedades: user y email.

A causa del ng-show, el span de color rojo es mostrado cuando el usuario es $dirty o $invalid.